

# DSGE: User Guide

Alfred Maußner

17 July 2010

## Contents

|                                               |           |
|-----------------------------------------------|-----------|
| <b>1. Introduction</b>                        | <b>1</b>  |
| <b>2. Installation</b>                        | <b>2</b>  |
| <b>3. Linear Approximation</b>                | <b>2</b>  |
| 3.1. Structure of the Model . . . . .         | 2         |
| 3.2. Example . . . . .                        | 3         |
| 3.3. Linearization . . . . .                  | 4         |
| 3.4. Solution Algorithm . . . . .             | 5         |
| 3.5. Additional Variables . . . . .           | 6         |
| <b>4. Quadratic Approximation</b>             | <b>6</b>  |
| 4.1. Structure of the Model . . . . .         | 6         |
| 4.2. Example . . . . .                        | 7         |
| 4.3. Derivatives . . . . .                    | 7         |
| 4.4. Solution Algorithm . . . . .             | 8         |
| <b>5. Output Options</b>                      | <b>8</b>  |
| <b>6. Summary of Global Control Variables</b> | <b>11</b> |
| <b>7. References</b>                          | <b>12</b> |

## 1. Introduction

The Gauss program DSGE provides a framework for solving DSGE models along the lines of Heer and Maußner (2009a), Chapter 2 and Heer and Maußner (2009b). Its aim is to allow less experienced users of Gauss to solve and simulate their model using either linear or quadratic approximations of the model's equilibrium conditions.

I assume that you are familiar with dynamic stochastic equilibrium models and, in particular, with perturbation methods that obtain approximate solutions of these models. I also assume that you know how to start the Gauss software under the Windows operating system and that you have a basic knowledge of the Gauss syntax.

In this document I use the typewriter font to print Gauss commands and file names. Vectors are typeset in boldface, e.g.,  $\mathbf{x}$  denotes a vector of  $n(x)$  elements.

## 2. Installation

The code of DSGE is stored in various files. The files with the `.src` extension contain source code that the user should not change. Gauss will look for these files in its `src` subdirectory. For instance, if `c:\gauss` is the directory where Gauss is installed then copy the files to this directory. You may also wish to use a different directory. In this case you must append the `src_path` variable in the file `Gauss.cfg` with the path to the directory where you want to store the `.src` files. Yet a third possibility is to copy the files in the working directory that you use for your project. This directory can be accessed via the working directory button or by executing the command `chdir path` from the Gauss command window, where `path` is the path to this directory.

The file `Tools.dll` must be placed in the Gauss `dlib` subdirectory (for instance in `c:\Gauss\dlib` if Gauss is installed in `c:\Gauss`). This dynamic link library contains routines written in Fortran that implement the Schur and the generalized Schur factorization of matrices. In order to use this library you also have to install the file `DFORRT.dll` in the `system32` subdirectory of your Windows 32 bit operating system (i.e. in `c:\windows\system32`) or in the `SysWOW64` subdirectory of your Windows 64 bit system.

The file `DSGE_?? .g`, where `??` either equals `LA` or `QA`, is designed as a shell, where you must input your model, determine the solution method and choose the output options. After you have input your code you should save this file under a different name so that you will be able to use the original file for other models.

The use of `DSGE_?? .g` is illustrated with the benchmark business cycle mode of Hansen (1985) as described in Heer and Maußner (2009a), Section 2.6.1. You should run these files to check your installation.

## 3. Linear Approximation

### 3.1. Structure of the Model

Solutions of DSGE models that build on a linearized version of the model are implemented in `DSGE_LA.g`. We distinguish between four types of variables:

- State variables: these are variables that have given initial conditions, as, for instance, the stock of capital or the level or nominal money balances. They are stored in the  $n(x)$  vector  $\mathbf{x}_t$ , where  $t$  denotes time.
- Costate variables: these are variables without initial conditions, as, for instance, the Lagrange multiplier of the household's budget constraint. They are stored in the  $n(\lambda)$  vector  $\boldsymbol{\lambda}_t$ .

- Control variable: these are variables without initial conditions that can be determined from the given values of the state and costate variable, as, for instance, output and working hours. They are stored in the  $n(u)$  vector  $\mathbf{u}_t$ .
- Shock variables: these are variables that follow given stochastic processes, as, for instance, the level of total factor productivity. They are stored in the  $n(z)$  vector  $\mathbf{z}_t$ .

Let  $\mathbb{E}_t$  denote expectations as of period  $t$ . The model can then be summarized by the  $n(u)$  equations

$$g^i(\mathbf{u}_t, \mathbf{x}_t, \boldsymbol{\lambda}_t, \mathbf{z}_t) = 0, \quad i = 1, 2, \dots, n(u), \quad (3.1)$$

the  $n(x) + n(\lambda)$  equations

$$\mathbb{E}_t h^i(\mathbf{u}_t, \mathbf{x}_t, \boldsymbol{\lambda}_t, \mathbf{z}_t, \mathbf{u}_{t+1}, \mathbf{x}_{t+1}, \boldsymbol{\lambda}_{t+1}, \mathbf{z}_{t+1}) = 0, \quad i = 1, 2, \dots, n(x) + n(\lambda), \quad (3.2)$$

and the system of  $n(z)$  equations

$$\mathbf{z}_{t+1} = \Pi \mathbf{z}_t + \boldsymbol{\epsilon}_{t+1}, \quad \boldsymbol{\epsilon} \sim N(\mathbf{0}_{n(z) \times 1}, \Sigma), \quad (3.3)$$

where  $N(\boldsymbol{\mu}, \Sigma)$  denotes the multivariate normal distribution with mean vector  $\boldsymbol{\mu}$  and covariance matrix  $\Sigma$ .

Usually the shocks in  $\mathbf{z}_t$  will not follow the vector autoregressive process (3.3), then we assume that the percentage deviations of the variables in  $\mathbf{z}_t$ ,  $z_{it}$ , from their unconditional means  $z_i$ , i.e.,

$$\hat{z}_{it} := \frac{z_{it} - z_i}{z_i}, \quad i = 1, 2, \dots, n(z)$$

are governed by (3.3).

### 3.2. Example

The stationary variables of the benchmark business cycle model from Section 1.5.1 of Heer and Maußner (2009a) are the capital stock per efficiency unit,  $k_t$ , output, consumption, and investment per efficiency unit,  $y_t$ ,  $c_t$ , and  $i_t$ , respectively, working hours  $N_t$ , the rental rate of capital  $r_t$ , the real wage per efficiency unit of labor  $w_t$ , the level of total factor productivity  $Z_t$ , and the scaled Lagrange multiplier of the household's budget constraint  $\lambda_t$ . Thus:

- $\mathbf{x}_t = k_t$ ,
- $\boldsymbol{\lambda}_t = \lambda_t$ ,
- $\mathbf{u}_t = [y_t, c_t, i_t, r_t, w_t, N_t]'$ ,

- $z_t = \ln Z_t$ .

The system of equations (3.1) is:

$$\begin{aligned}
0 &= c_t^{-\eta}(1 - N_t)^{\theta(1-\eta)-\lambda_t}, \\
0 &= c_t^{1-\eta}(1 - N_t)^{\theta(1-\eta)-1} - \lambda_t w_t, \\
0 &= r_t - \alpha Z_t N_t^{1-\alpha} k_t^{\alpha-1}, \\
0 &= w_t - (1 - \alpha) Z_t N_t^{-\alpha} k_t^\alpha, \\
0 &= y_t - Z_t N_t^{1-\alpha} k_t^\alpha, \\
0 &= y_t - i_t - c_t,
\end{aligned}$$

where  $\eta$ ,  $\theta$ , and  $\alpha$  are parameters of the model. These equations determine the variables in  $\mathbf{u}_t$  given  $k_t$ ,  $\lambda_t$  and  $Z_t$ . The model's dynamics from (3.2) is governed by

$$\begin{aligned}
0 &= a k_{t+1} - (1 - \delta) k_t - i_t, \\
0 &= \lambda_t - \beta a^{-\eta} \mathbb{E}_t \lambda_{t+1} (1 - \delta + r_{t+1}),
\end{aligned}$$

where  $a$  and  $\delta$  are also parameters of the model. Finally, with  $\mathbb{E} Z_t = Z \equiv 1$ ,  $z_t := \ln Z_t \simeq ((Z_t) - 1)/1$  evolves according to

$$z_{t+1} = \varrho z_t + \epsilon_{t+1}, \quad \epsilon \sim N(0, \sigma).$$

### 3.3. Linearization

The model in (3.1) and (3.2) is linearized at the stationary solution of the deterministic counterpart of the model, which is obtained from (3.1) and (3.2) in the following steps:

Step1: Replace  $\mathbf{z}_t$  by its unconditional mean  $\mathbf{z}$  so that the expectations operator  $\mathbb{E}_t$  can be ignored.

Step2: Assume stationarity, i.e., drop the time index from all variables.

Step3: Solve

$$\begin{aligned}
0 &= g^i(\mathbf{u}, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{z}), & i &= 1, 2, \dots, n(u), \\
0 &= h^i(\mathbf{u}, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{z}, \mathbf{u}, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{z}), & i &= 1, 2, \dots, n(x) + n(\lambda).
\end{aligned}$$

for  $\mathbf{u}$ ,  $\mathbf{x}$ , and  $\boldsymbol{\lambda}$ .

Let  $\hat{x}_t := (x_t - x)/x$  denote the percentage deviation of a variable from its stationary solution  $x$ . In terms of these variables the log-linearized model can be written as:

$$C_u \hat{\mathbf{u}}_t = C_{x\lambda} \begin{bmatrix} \hat{\mathbf{x}}_t \\ \hat{\boldsymbol{\lambda}}_t \end{bmatrix} + C_z \hat{\mathbf{z}}_t, \quad (3.4a)$$

$$D_{x\lambda} \mathbb{E}_t \begin{bmatrix} \hat{\mathbf{x}}_{t+1} \\ \hat{\boldsymbol{\lambda}}_{t+1} \end{bmatrix} + F_{x\lambda} \begin{bmatrix} \hat{\mathbf{x}}_t \\ \hat{\boldsymbol{\lambda}}_t \end{bmatrix} = D_u \mathbb{E}_t \hat{\mathbf{u}}_{t+1} + F_u \hat{\mathbf{u}}_t + D_z \mathbb{E}_t \hat{\mathbf{z}}_{t+1} + F_z \hat{\mathbf{z}}_t, \quad (3.4b)$$

$$\hat{\mathbf{z}}_t = \Pi \hat{\mathbf{z}}_{t-1} + \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim N(\mathbf{0}, \Sigma) \quad (3.4c)$$

You can either derive this structure using paper and pencil or input the equations (3.1) and (3.2) and let the program determine the matrices in (3.4a) and (3.4b). In the first case you must set the variable `_equations` equal to zero and input the non-zero elements of the matrices  $C_u$ ,  $C_{x\lambda}$ ,  $C_z$ ,  $D_{x\lambda}$ ,  $F_{x\lambda}$ ,  $D_u$ ,  $F_u$ ,  $D_z$ , and  $F_z$ . In the second case put `_equations=1` and supply the equations (3.1) and (3.2) in the Gauss procedures `Sys1` and `Sys2`, respectively. Note that  $df(\ln x) = f_x \hat{x}$ . Therefore, the input to `Sys1` and `Sys2` are the (natural) logarithms of the variables not the levels of the variables. Therefore, `Sys1` and `Sys2` must contain the statement `w=exp(w)`, where  $\mathbf{w} = [\mathbf{u}, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{z}]$ .

As a final step you must input the matrices  $\Pi$  and  $\Sigma^{1/2}$  from equation (3.4c).

### 3.4. Solution Algorithm

Internally the program reduces (3.4) to the system

$$B\mathbb{E}_t \begin{bmatrix} \hat{\mathbf{x}}_{t+1} \\ \hat{\boldsymbol{\lambda}}_{t+1} \end{bmatrix} = A \begin{bmatrix} \hat{\mathbf{x}}_t \\ \hat{\boldsymbol{\lambda}}_t \end{bmatrix} + C\hat{\mathbf{z}}_t, \quad (3.5)$$

where

$$\begin{aligned} B &:= (D_{x\lambda} - D_u C_u^{-1} C_{x\lambda}), \\ A &:= -(F_{x\lambda} - F_u C_u^{-1} C_{x\lambda}), \\ C &:= (D_z + D_u C_u^{-1} C_z) \Pi + (F_z + F_u C_u^{-1} C_z). \end{aligned}$$

If the matrix  $B$  is invertible, which will usually be the case if you correctly specify (3.1) and (3.2), you can use the Schur factorization of  $W = B^{-1}A$  to obtain linear feed back rules of the form:

$$\begin{aligned} \hat{\mathbf{x}}_{t+1} &= L_x^x \hat{\mathbf{x}}_t + L_z^x \hat{\mathbf{z}}_t, \\ \hat{\mathbf{u}}_t &= L_x^u \hat{\mathbf{x}}_t + L_z^u \hat{\mathbf{z}}_t, \\ \hat{\boldsymbol{\lambda}}_t &= L_x^\lambda \hat{\mathbf{x}}_t + L_z^\lambda \hat{\mathbf{z}}_t. \end{aligned} \quad (3.6)$$

In this case, set the global control variable `_GSchur` equal to zero. If  $B$  is not invertible (something you will at least get to know if the program stops with an error message) – and if you are sure that your matrices or your equations are correctly specified – you can use the generalized Schur factorization to compute the feed back rules (3.6).

The program will be able to compute a solution if  $n(x)$  of the eigenvalues of  $B^{-1}A$  or of the matrix pencil  $(B, A)$  are within the unit circle and  $n(\lambda)$  of the eigenvalues are outside the unit circle. If this condition is not met the program stops with an error message. You will also get error messages if

- the program is not able to obtain a factorization of  $(B, A)$ ,
- if the program detects complex coefficients in  $L_x^x$ ,  $L_z^x$ ,  $L_x^\lambda$ , or  $L_z^\lambda$ .

In both cases the computed solution will be misleading and you should check your program.

### 3.5. Additional Variables

Occasionally you may want solutions for variables that can not be included in  $\mathbf{u}_t$  or  $\boldsymbol{\lambda}_t$ . For instance, in the benchmark model consider  $\mathbb{E}_t \lambda_{t+1}$ . From the linear solution

$$\hat{\lambda}_t = L_x^\lambda \hat{x}_t + L_z^\lambda \hat{z}_t$$

you get

$$\begin{aligned} \mathbb{E}_t \hat{\lambda}_{t+1} &= \mathbb{E}_t \{ L_x^\lambda \hat{x}_{t+1} + L_z^\lambda \hat{z}_{t+1} \}, \\ &= L_x^\lambda \{ L_x^x \hat{x}_t + L_z^x \hat{z}_t \} + L_z^\lambda \mathbb{E}_t \{ \Pi \hat{z}_t + \epsilon_{t+1} \}, \\ &= L_x^\lambda L_x^x \hat{x}_t + (L_x^\lambda L_z^x + L_z^\lambda \Pi) \hat{z}_t. \end{aligned}$$

Therefore, you can determine  $\mathbb{E}_t \hat{\lambda}_{t+1}$  as a linear function of  $\hat{x}_t$  and  $\hat{z}_t$  with matrices

$$\begin{aligned} L_x^y &= L_x^\lambda L_x^x, \\ L_z^y &= L_x^\lambda L_z^x + L_z^\lambda \Pi. \end{aligned}$$

Note, that the program must compute the linear solution before it can determine  $L_x^y$  and  $L_z^y$ . For this reason you must input the two matrices after the statement

```
#include SolveLA2.src;
```

in the file DSGE\_LA.

## 4. Quadratic Approximation

### 4.1. Structure of the Model

The quadratic approximate solution of the model is implemented in DSGE\_QA. Since it rests on numeric second derivatives you always have to specify the equations of your model. The program distinguishes between three types of variables:

- State variables (see above) are collected in the vector  $\mathbf{x}_t$ ,
- variables without given initial conditions are collected in the vector  $\mathbf{y}_t$ ,
- and the model's shocks in the vector  $\mathbf{z}_t$ .

The model is summarized in the  $n(x) + n(y)$  equations

$$\mathbb{E}_t g^i(\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t, \mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}) = 0, \quad i = 1, 2, \dots, n(x) + n(y) \quad (4.1)$$

and the transition equation for the shocks

$$\begin{aligned} \mathbf{z}_{t+1} &= \Pi \mathbf{z}_t + \boldsymbol{\epsilon}_{t+1}, \\ \boldsymbol{\epsilon}_{t+1} &= \sigma \Omega \boldsymbol{\eta}_{t+1}, \quad \boldsymbol{\eta}_{t+1} \sim N(\mathbf{0}_{n(z)}, I_{n(z)}). \end{aligned} \quad (4.2)$$

Note that (4.2) implies that the mean of  $\mathbf{z}$  is  $\mathbf{0}_{n(z)}$ . Thus, if the shocks in your model have mean  $\mathbf{z} \neq \mathbf{0}_{n(z)}$  you must replace  $\mathbf{z}_t$  by  $\hat{\mathbf{z}}_t$  in equations (4.1) and (4.2).

## 4.2. Example

In order to apply the quadratic solution to the benchmark business cycle model we collect its variables in

- $\mathbf{x}_t = k_t$ ,
- $\mathbf{y}_t = [y_t, c_t, i_t, r_t, w_t, N_t, \lambda_t]'$ ,
- $z_t = \ln Z_t$ .

The system of equations (4.1) is:

$$\begin{aligned}
0 &= c_t^{-\eta} (1 - N_t)^{\theta(1-\eta)-\lambda_t}, \\
0 &= c_t^{1-\eta} (1 - N_t)^{\theta(1-\eta)-1} - \lambda_t w_t, \\
0 &= r_t - \alpha Z_t N_t^{1-\alpha} k_t^{\alpha-1}, \\
0 &= w_t - (1 - \alpha) Z_t N_t^{-\alpha} k_t^{\alpha}, \\
0 &= y_t - Z_t N_t^{1-\alpha} k_t^{\alpha}, \\
0 &= y_t - i_t - c_t, \\
0 &= a k_{t+1} - (1 - \delta) k_t - i_t, \\
0 &= \lambda_t - \beta a^{-\eta} \mathbb{E}_t (1 - \delta + r_{t+1}).
\end{aligned}$$

The parameters of (4.2) are  $\varrho$ ,  $\sigma$  and  $\Omega = 1$ .

## 4.3. Derivatives

The program computes first and second derivatives of the system (4.1) at  $(\mathbf{x}, \mathbf{y}, \mathbf{z})$  that solve

$$g^i(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{x}, \mathbf{y}, \mathbf{z}) = 0, \quad i = 1, 2, \dots, n(x) + n(y).$$

In `DSGE_QA` you must provide both the system of equations (4.1) and the stationary solution of the model. The stationary values must be stored in the vectors `_xstar`, `_ystar`, and `_zstar`. Remember that  $\mathbf{z} = \mathbf{0}_{n(z) \times 1}$  so that `_zstar` always has to be a vector of  $n(z)$  zero elements.

The second derivatives are computed for each equation in (4.1). To accomplish this the procedure `SYS` must contain the command

```
if _eqno==0; retp(fx); else; retp(fx[_eqno]); endif;
```

before the `endp` statement. Different from `DSGE_LA` the procedure `SYS` receives  $\mathbf{w} = (\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{x}, \mathbf{y}, \mathbf{z})$  as argument.

## 4.4. Solution Algorithm

The first step in the computation of the solution is similar to the linear approximation. For this reason you can use `DSGE_QA` also to obtain a linear approximate solution of your model. To accomplish this set the global control variable `_linear` equal to 1. From the Jacobian matrix of (4.1) the program derives

$$B\mathbb{E}_t \begin{bmatrix} \bar{\mathbf{x}}_{t+1} \\ \bar{\boldsymbol{\lambda}}_{t+1} \end{bmatrix} = A \begin{bmatrix} \bar{\mathbf{x}}_t \\ \bar{\boldsymbol{\lambda}}_t \end{bmatrix} + C\mathbf{z}_t, \quad (4.3)$$

where  $\bar{\mathbf{x}}_t = \mathbf{x}_t - \bar{\mathbf{x}}$  and  $\bar{\boldsymbol{\lambda}}_t = \boldsymbol{\lambda}_t - \bar{\boldsymbol{\lambda}}$  ( $\bar{\mathbf{z}}_t$  is always equal to  $\mathbf{z}_t$ ). The matrices  $A$ ,  $B$ , and  $C$  are functions of the model's parameters. The linear part of the feed back rules is computed via the generalized Schur factorization of  $(B, A)$ . Therefore, you will get the same error messages as in the linear case if the factorization fails, if the conditions on the eigenvalues are not satisfied or if complex coefficients appear in the solution.

The quadratic part of the feed back rule for  $x_{it+1}$  or  $y_{it}$  is given by

$$q_t^i := \frac{1}{2} \begin{bmatrix} \bar{\mathbf{x}}_t' & \mathbf{z}_t' & \sigma \end{bmatrix} H^i \begin{bmatrix} \bar{\mathbf{x}}_t \\ \mathbf{z}_t \\ \sigma \end{bmatrix}$$

where the matrix  $H^i$  depends on the parameters of the first and second derivatives of (4.1). Again, if  $\hat{\mathbf{z}}_t$  instead of  $\mathbf{z}_t$  is governed by (4.2) then  $\hat{\mathbf{z}}_t$  is an argument of the quadratic feed back rule.

## 5. Output Options

The program provides two output options:

- if `_IR=1` impulse responses will be computed and plotted,
- if `_Moments=1` second moments obtained from simulations of the model will be computed and printed to the file you specify in the string on the right of the command `outfile="summary.txt"`.
- if `_confidence=1` the program computes and prints 95 percent intervals for the simulated moments.

For instance, the program's graphic output for the benchmark business cycle model may look like Figure 5.1 and the printed output could be as presented in Table 5.1.

The number of periods considered in the computation of impulse responses is stored in the variable `nobs1`. Second moments are computed as average over `nofs` simulations of the model, where each simulation computes times series of length `nobs`. The

Table 5.1

---

Second moments from simulated data:

|                        |      |      |      |      |
|------------------------|------|------|------|------|
| Output                 | 1.44 | 1.00 | 1.00 | 0.65 |
| Consumption            | 0.56 | 0.39 | 0.99 | 0.66 |
| Investment             | 6.13 | 4.26 | 1.00 | 0.64 |
| Hours                  | 0.77 | 0.54 | 1.00 | 0.64 |
| Real Wage              | 0.67 | 0.47 | 0.99 | 0.66 |
| Rental Rate of Capital | 1.48 | 1.03 | 0.98 | 0.64 |

Column 1: Variable name

Column 2: Standard deviation

Column 3: Standard deviation relative to standard deviation of Output

Column 4: Cross correlation with Output

Column 5: First order autocorrelation

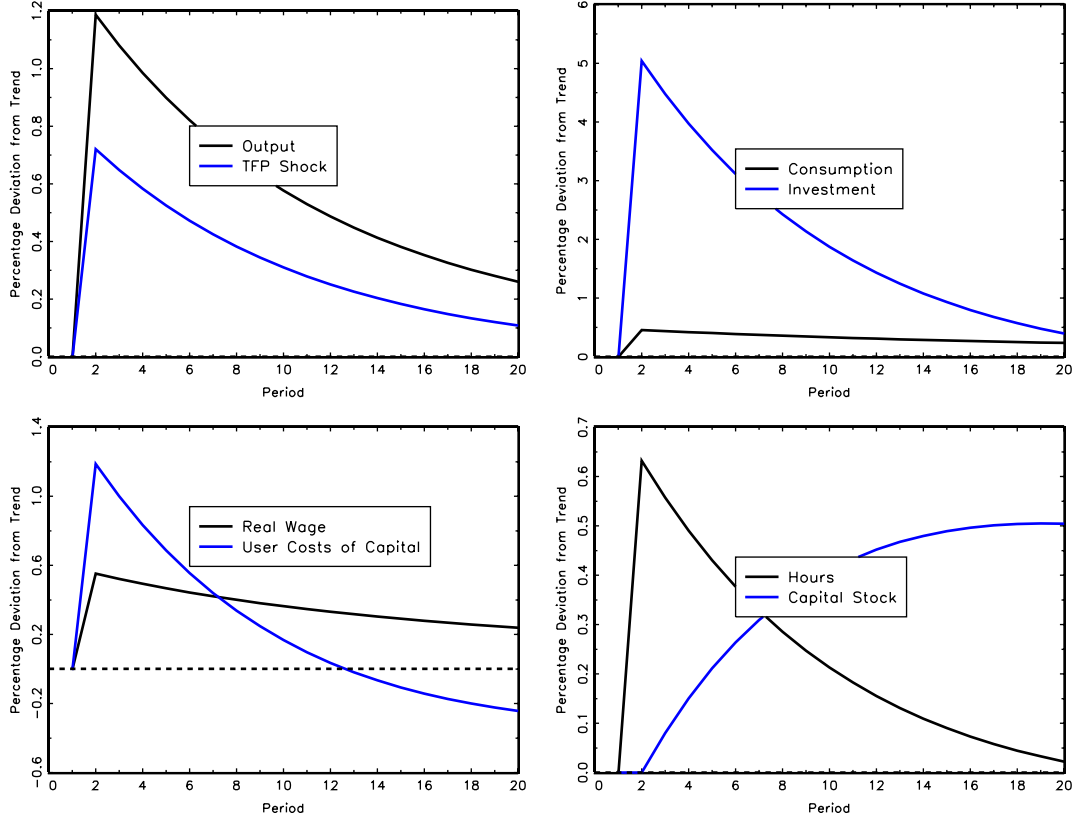
---

simulations rest on draws of random numbers. If `_loadShocks=0`, the program draws pseudo normal distributed random numbers for each simulation. If you want to compare different versions of your model you may want to use the same set of random numbers in order not to confuse systematic differences with random ones. In this case use `_loadShocks=1`. The program will look for a file `eps_array.fmt` in the current working directory. To create this file for your settings of `nobs`, `nofs` and `_nz` (the number of shocks in your model) uses `_MakeShocks=1` once and set this variable equal to 0 in all other runs of your program.

If required, you can pass the time series through the Hodrick-Prescott filter (see Heer and Maußner (2009), Section 12.4). In this case, assign the filter weight (e.g. 1600) to the variable `_HPL`. If `_HPL=0`, the program does not filter the data.

Impulse responses are computed and plotted for each element of  $\mathbf{z}_t$ . The graphic files that store the plots are saved in files named `Figure?.tkf`, where `?` is the number of the element  $z_i$ ,  $i = 1, 2, \dots, n(z)$ , in the current working directory of Gauss. Each plot can consist of up to six panels (graphic windows) in which you can plot the impulse responses of the variables of your model (see Figure 5.1). There are two other options that determine the way impulse responses are plotted. If `_scale=0`, each graphics window will be scaled individually. For `_scale=1` the same scale will be applied to all panels. Impulse responses will be colored if `_color=1`, otherwise black lines will be used.

Figure 5.1  
Impulse Responses from the Benchmark Business Cycle Model



The output of the program is further controlled by a Gauss structure. There are as many instances of this structure as there are variables in your model. For each variable for which you want to obtain either graphics or printed output your program must contain the following lines of code:

```

1 Var[1].varname="Output";
2 Var[1].vartype="u";
3 Var[1].varpos=5;
4 Var[1].varprint=1;
5 Var[1].relsx=1;
6 Var[1].crosscorr=1;
7 Var[1].varplot=1;
8 Var[1].plotno=1;

```

In this example, `Var[1]` refers to the first instance of the structure `Var`. This instance stores information on the variable named `Output`. This variable is a member of the vector  $\mathbf{u}_t$ . In `DSGE_LA` the structure member `vartype` can be assigned the strings `u`, `x`, `1` (if the variable belongs to the vector  $\boldsymbol{\lambda}_t$ ), `z`, or `y`. In `DSGE_QA` accepted strings are `x`, `y`, and `z`. In the example, `output` is the fifth variable in the vector  $\mathbf{u}$  (see line

3). In lines 4-7, the assignment of 1 stands for true and 0 for false. Thus, `varprint=1` tells the program that you want information on the second moments of the variable `output` to be printed to the file specified in the variable `outfile`. Without any further information the program prints the standard deviation of this variable and its first-order autocorrelation. If you also set `relsx` and `crosscorr` to true the output table has two additional columns that show the standard deviations of other variables relative to the standard deviation of `output` and the cross-correlations of other variables with `output`.

In line 7 and 8 you tell the program to print the impulse response of `output` to the graphics panel 1. Accepted values for `plotno` are the integers 1 through 6.

Finally, you may want to overwrite the program's automatic positioning of plot legends. In this case, you must edit the global variable `_plegend`. If you want to change the lower left corner of the legend in panel  $i$  of figure  $j$  to the plot coordinates  $(x, y)$  type

```
1 _plegend[j,i,1]=x;
2 _plegend[j,i,2]=y;
```

## 6. Summary of Global Control Variables

| File            | Variable                 | Possible Settings and Effect                                                                                                      |
|-----------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| DSGE_LA         | <code>_GSchur</code>     | 1: the generalized Schur factorization will be used                                                                               |
| DSGE_LA         | <code>_equations</code>  | 1: the program expects the equations of your model in <code>Sys1</code> and <code>Sys2</code>                                     |
| DSGE_QA         | <code>_linear</code>     | 1: the program computes only linear feed back rules                                                                               |
| DSGE_LA/DSGE_QA | <code>_Moments</code>    | 1: the program computes second moments                                                                                            |
| DSGE_LA/DSGE_QA | <code>_confidence</code> | 1: the program computes and prints 95 percent intervals for the simulated moments                                                 |
| DSGE_LA/DSGE_QA | <code>_IR</code>         | 1: the program computes and plots impulse responses                                                                               |
| DSGE_LA/DSGE_QA | <code>_nobs1</code>      | the number of periods in each impulse response                                                                                    |
| DSGE_LA/DSGE_QA | <code>_nobs</code>       | the length of simulated time series                                                                                               |
| DSGE_LA/DSGE_QA | <code>_nofs</code>       | the number of simulations                                                                                                         |
| DSGE_LA/DSGE_QA | <code>_HPL</code>        | the filter weight of the HP-filter, if equal to 0 the HP-filter will not be applied                                               |
| DSGE_LA/DSGE_QA | <code>_scale</code>      | 1: all graphic panels share the same scale                                                                                        |
| DSGE_LA/DSGE_QA | <code>_color</code>      | 1: colored impulse responses are plotted                                                                                          |
| DSGE_LA/DSGE_QA | <code>_LoadShocks</code> | 1: the program uses random numbers from the file <code>eps_array.fmt</code>                                                       |
| DSGE_LA/DSGE_QA | <code>_MakeShocks</code> | 1: the program draws random numbers and stores them in the file <code>eps_array.fmt</code> in the current Gauss working directory |
| DSGE_LA/DSGE_QA | <code>_outfile</code>    | string variable, the name of the output file                                                                                      |

## 7. References

- Hansen, Gary D. 1985. Indivisible Labor and the Business Cycle, *Journal of Monetary Economics*. Vol. 16. 309-327.
- Heer, Burkhard und Alfred Maußner. 2009a. Dynamic General Equilibrium Modeling. Springer: Berlin
- Heer, Burkhard und Alfred Maußner. 2009b. Computation of Business-Cycle Models with the Generalized Schur Method. *Indian Growth and Development Review*. Vol. 2. 173-182